Math. Prog. Comp. (2010) 2:231–257
DOI 10.1007/s12532-010-0018-0

FULL LENGTH PAPER

A heuristic to generate rank-1 GMI cuts

Sanjeeb Dash · Marcos Goycoolea

Received: 7 December 2009 / Accepted: 16 September 2010 / Published online: 16 November 2010

Abstract Gomory mixed-integer (GMI) cuts are among the most effective cutting planes for general mixed-integer programs (MIP). They are traditionally generated from an optimal basis of a linear programming (LP) relaxation of a MIP. In this paper we propose a heuristic to generate useful GMI cuts from additional bases of the initial LP relaxation. The cuts we generate have rank one, i.e., they do not use previously generated GMI cuts. We demonstrate that for problems in MIPLIB 3.0 and MIPLIB 2003, the cuts we generate form an important subclass of all rank-1 mixed-integer rounding cuts. Further, we use our heuristic to generate globally valid rank-1 GMI cuts at nodes of a branch-and-cut tree and use these cuts to solve a difficult problem from MIPLIB 2003, namely *timtab2*, without using problem-specific cuts.

Mathematics Subject Classification (2000) 90C11 · 90C57

1 Introduction

Gomory mixed-integer (GMI) cutting planes (or *cuts*) [28], form one of the most important classes of cutting planes for solving general mixed-integer programs (MIP). In the 1970s, Geoffrion and Graves [27] combined GMI cuts with linear programming

M. Goycoolea was supported by FONDECYT grant 11075028 and ANILLO grant ACT-88.

S. Dash (✉)
Mathematical Programming Group, IBM T. J. Watson Research Center,
Yorktown Heights, NY, USA
e-mail: sanjeebd@us.ibm.com

M. Goycoolea
School of Business, Universidad Adolfo Ibáñez, Santiago, Chile
e-mail: marcos.goycoolea@uai.cl

(LP) based branch-and-bound to solve a MIP arising in a practical application, but did not systematically study the effect of GMI cuts on a range of MIPs. Balas et al. [6] first showed how to use GMI cuts in a generally effective manner in the 1990s, and the usefulness of GMI cuts in solving practical MIPs is confirmed in [11,13].

Following these papers, GMI cuts are typically generated from rows of an optimal simplex tableau associated with an LP relaxation of a MIP, usually in a small number of rounds. A *round* [6] of GMI cuts consists of generating GMI cuts from multiple rows of an LP relaxation tableau and augmenting the relaxation with the cuts violated by the basic solution defined by the tableau. The effectiveness of rounds of GMI cuts is partially explained by the papers of Dash and Günluk [20], Fischetti and Saturni [25], and Fukasawa and Goycoolea [26], who essentially show that no additional cuts from the rows of an optimal simplex tableau are useful over and above GMI cuts from these rows for MIPLIB 3.0 [12] problems. However, after many successive rounds of cuts from optimal tableau rows, the GMI cuts are often very dense or have large variation in coefficient magnitudes, thus leading to hard-to-solve LP relaxations when they are added. Further, because of floating-point computations with limited accuracy, the computed cuts are often invalid [32]. These factors inhibit extensive GMI cut generation.

GMI cuts (as described above) are equivalent to mixed integer rounding (MIR) inequalities [31] generated from simplex tableau rows. We say that a valid inequality or cutting plane has MIR rank 1 (abbreviated as rank 1) if it can be derived as a MIR cut from some linear combination of the rows of a MIP (which has rank 0). A valid inequality has rank $k > 0$ if it can be derived as a MIR cut from inequalities with rank $k - 1$ or less, but cannot be derived from inequalities with rank $k - 2$ or less. GMI cuts from a simplex tableau of the initial LP relaxation of a MIP have rank 1. Multiple rounds of GMI cuts may lead to cuts with rank 2 or more.

Marchand and Wolsey [30] showed that MIR cuts (different from GMI cuts) are useful in solving general MIPs. Fischetti and Lodi [24] obtained strong lower bounds for the pure integer programs (with an objective function to be minimized) in MIPLIB 3.0 by optimizing over the Chvátal closure, i.e., by iteratively generating violated rank-1 Gomory-Chvátal cuts. Balas and Saxena [8], and Dash et al. [22] later obtained significantly stronger lower bounds for MIPLIB 3.0 instances by approximately optimizing over the MIR closure (the set of points satisfying all rank-1 MIR cuts), via iterative generation of violated rank-1 MIR cuts. However, for an arbitrary point, finding a violated MIR cut is NP-hard [15]. In [8,24] and [22], an auxiliary MIP is solved to find a linear combination of rows yielding a violated cut, and this process is very time-consuming.

Our work is inspired by the fact that aggressive generation of rank-1 MIR cuts allows one to close a large fraction of the integrality gap for MIPLIB 3.0 problems [8,22], and by the fact that the first round of GMI cuts (which have rank 1) can be obtained very quickly, has appealing numerical properties [6,20,25,26], and closes a non-trivial fraction of the integrality gap closed by all rank-1 MIR cuts (about 26.09% for the first round of GMI cuts, and 76.52% for rank-1 MIR cuts, see Table 4). We study the following two questions in this paper. (A) Do rank-1 GMI cuts define a good approximation to the set of rank-1 MIR cuts for practical MIPs? (B) Can one design a fast algorithm to find violated rank-1 GMI cuts? It is shown in [19] that the MIR closure is strictly contained in the rank-1 GMI cut closure for some MIP, though

this does not answer Question A. As for the second question, we do not know if it is NP-hard to separate over the family of rank-1 GMI cuts. Partial answers to these questions are given in Bonami and Minoux [10] discussed below.

We give positive answers to these questions for MIPLIB 3.0 problems via a fast heuristic to find rank-1 GMI cuts. Given a MIP with LP relaxation $L := \min\{cx : Ax = b, x \geq 0\}$ and a non-basic solution x^* of L to be separated, our method explores bases with properties similar to an optimal basis of L, namely (1) the basic columns are contained in the support of the point to be separated (assuming non-degeneracy), and (2) the nonbasic columns correspond to variables with values close to zero. For every basis we find, we add all GMI cuts violated by x^*. If A is sparse and has m rows, and x^* is non-basic with $m + t$ nonzeros for small t, then (1) our method often finds a basis of L such that some of the associated GMI cuts are violated by x^*; (2) the time to find such a basis is comparable to the time for a round of GMI cuts derived from an optimal basis of L; (3) such GMI cuts close a much larger fraction of the integrality gap than the first round of GMI cuts. We iterate this process, and generate multiple rounds of rank-1 GMI cuts.

Our method is much faster than the MIP-based separation methods of Balas and Saxena, and of Dash et al., but is not much weaker for MIPLIB 3.0 instances in terms of the integrality gap closed. For 54 out of the 65 instances in MIPLIB 3.0 our default (resp., expensive) heuristic closes 62.16% (resp., 64.58%) of the integrality gap, on average; the corresponding numbers for rank-1 MIR cuts in Dash et al. [22] and Balas and Saxena [8] are 62.53 and 76.52%, respectively (Table 4). One round of GMI cuts closes 26.09% of the integrality gap. Further, for the MIPs in MIPLIB 2003 [1] not contained in MIPLIB 3.0, our default heuristic closes 39.68% of the integrality gap, as opposed to 18.37% if one round of GMI cuts is used.

In related work, Balas and Perregard [7] showed, for 0–1 problems, that *strengthened lift-and-project* (L&P) cuts (defined in [5]) are the same as rank-1 GMI cuts. They showed that an L&P cut from a basic feasible solution (say μ) of the cut generation LP (CGLP) associated with L&P cuts is equivalent to an L&P cut for some tableau row (say r), and the strengthened L&P cut from μ is just the GMI cut from r. To separate a basic solution x^*, they give a fast algorithm to find r, which starts with a row of the simplex tableau defining x^* and performs pivoting steps. Their implementation of this algorithm and that of Balas and Bonami [4] yields good results for MIPLIB instances. In contrast with our work, they use rank-1 GMI cuts from non-optimal tableaus to strengthen GMI cuts from the optimal tableau, and not as a source of additional rank-1 cuts. In particular, after one round of GMI cuts, they generate potentially higher rank cuts. For other ways of generating GMI cuts, see Ceria et al. [16], and Andersen et al. [2].

For 0–1 problems, Bonami and Minoux [10] find L&P cuts cutting off a point x^*, and replace these cuts with strengthened L&P cuts. They thereby approximately optimize over $P_{L\&P+S}$—the points satisfying all strengthened L&P cuts, which equals the rank-1 GMI cut closure by the results of Balas and Perregaard above. Their results for 0–1 MIPLIB 3.0 instances (23 out of the 54 we study) are fairly good. On the average, they close 52.36% of the integrality gap, as opposed to 19.91% gap closed with one round of GMI cuts. Our default heuristic closes 65.27% of the integrality gap; their computing times have the same order of magnitude as ours.

The effect of multiple rounds of GMI cuts has been studied earlier (e.g., by Balas et. al. [6], and in [18]); however the GMI cuts are not restricted to have rank 1. Our method avoids some of the numerical problems associated with GMI cuts in these papers, allowing us to generate many rounds of rank-1 GMI cuts. Further, our heuristics yield globally valid rank-1 GMI cuts at the nodes of a branch-and-cut tree, even for problems with general integer variables. In contrast, in Balas et. al. [6], the technique to obtain a globally valid GMI cut at a node of a branch-and-cut tree by lifting applies only to 0–1 mixed integer programs. We show that the cuts we generate significantly reduce the size of the branch-and-cut tree for some MIPLIB problems. In particular, we can solve *timtab2* from MIPLIB 2003, a problem which was previously solved only using problem-specific cuts (see [14,29]).

The paper is structured as follows. In Sect. 2 we discuss the standard way of generating GMI cuts described in the literature. In Sect. 3 we describe the main idea behind our method of choosing suitable non-optimal bases to generate GMI cuts. In Sect. 4, we discuss how to incorporate our cutting plane technique in a branch-and-cut setting, and also how to use branch-and-bound to find suitable bases for generating GMI cuts. We present our computational results in Sect. 5. Finally, we conclude in Sect. 6 with a discussion on the limitations of and possible improvements to our technique. We also briefly describe how our techniques can be used in the context of nonlinear programming problems.

2 The GMI cut

Consider a general MIP given in the form

$$M := \min\{cx : x \in \mathbb{R}^n_+,\ Ax = b,\ x_i \in \mathbb{Z}\ \forall i \in S\}, \tag{1}$$

where $S \subseteq \{1, \ldots, n\}$. We assume A has m rows and has full row rank, i.e., the rank of A is m, and therefore, $m \leq n$. We assume all data is rational. For a number $t \in \mathbb{R}$, we define $\hat{t} = t - \lfloor t \rfloor$.

Assume that the equation

$$\sum_{i=1}^{n} a_i x_i = \beta \tag{2}$$

is implied by $Ax = b$. Further assume that β is not integral, i.e., $\hat{\beta} = \beta - \lfloor \beta \rfloor \neq 0$. The Gomory mixed-integer (GMI) cutting plane or inequality for (2) is

$$\sum_{i \in S, \hat{a}_i < \hat{\beta}} \frac{\hat{a}_i}{\hat{\beta}} x_i + \sum_{i \in S, \hat{a}_i \geq \hat{\beta}} \frac{1 - \hat{a}_i}{1 - \hat{\beta}} x_i + \frac{1}{\hat{\beta}} \sum_{i \notin S, a_i > 0} a_i x_i - \frac{1}{1 - \hat{\beta}} \sum_{i \notin S, a_i < 0} a_i x_i \geq 1, \tag{3}$$

and is satisfied by the solutions of M. We refer to the equation $\sum_{i=1}^{n} a_i x_i = \beta$ as the *base constraint* of (3). Note that every integer variable x_i such that a_i is integral gets a coefficient of 0 in (3) because $\hat{a}_i = 0$.

Let L_1 be the linear program

$$L_1 := \min\{cx : Ax = b, x \geq 0\}, \tag{4}$$

the standard LP relaxation of (1). An $m \times m$ sub-matrix B of linearly independent columns in A is referred to as a *basis* (or basis matrix by others) of A, or a basis of L_1. If B is a basis of A, then A can be written as (B, N) where N is the set of remaining columns. Let $\mathcal{I} = \{1, \ldots, n\}$ and let $\mathcal{I}_B \subseteq \mathcal{I}$ be the set of indices of columns in B. Let $\mathcal{I}_N = \mathcal{I} \setminus \mathcal{I}_B$ be the set of indices of all other columns. The variables x_j with $j \in \mathcal{I}_B$ are called *basic* and all others *non-basic* with respect to B. We define x_B (resp. x_N) to be the vector of variables such that the ith variable in x_B (resp. x_N) corresponds to the ith column in B (resp. N).

Given a basis B of A, the system of equations $Ax = b$ is equivalent to the system

$$x_B + B^{-1}Nx_N = B^{-1}b. \tag{5}$$

This is the *tableau* associated with B. Each row of system (5) is known as a *tableau row*. We say that a solution x^* of $Ax = b$ is *basic* with respect to B if $x^*_N = 0$ and $x^*_B = B^{-1}b$. We say that a basic solution x^* is *feasible* if $x^*_B \geq 0$. If L_1 has an optimal solution, then it has an optimal solution which is basic for some basis B.

Tableaus and basic solutions are important in the context of GMI cuts. Observe that the kth row of the tableau (5) is of the form

$$x_j + \sum_{i \in \mathcal{I}_N} \bar{a}_i x_i = \bar{\beta} \tag{6}$$

where x_j is the kth basic variable, $\bar{a}$ is the kth row of $B^{-1}N$, and $\bar{\beta} = (B^{-1}b)_k$. If x^* is a feasible solution of (4) which is basic with respect to B, then $x^*_i = 0$ for all $i \in \mathcal{I}_N$ and $x^*_j = \bar{\beta}$. Thus, the GMI cut with base inequality (6) is violated by x^* if $j \in S$ and x^*_j is not integral. We say that the GMI cuts with base inequality (6) for $k = 1, \ldots, m$ are based on B.

We refer to the step of adding the violated GMI cuts based on a basis B simultaneously as a *round* of GMI cuts, following Balas et. al. [6]. They showed that adding GMI cuts in rounds is very effective for MIPLIB 3.0 [12] problems. In [6, Fig. 1], the authors demonstrate that adding all violated GMI cuts based on an optimal basis is more effective than adding a fraction of the violated cuts. From now on, we will assume a round consists of all violated GMI cuts. Recent work in [20,25], and [26] provides additional evidence of the value of adding GMI cuts in rounds for MIPLIB 3.0 problems. In [20] the authors show that an optimal solution of the relaxation obtained by adding all GMI cuts based on an optimal basis B of L_1 satisfies all group cuts (interpolated group cuts in [25]) derived from individual tableau rows of L_1 corresponding to B. In [26], Fukasawa and Goycoolea show that adding knapsack cuts based on rows of the optimal tableau of L_1 (i.e., cuts which use both upper and lower bounds on the variables) does not yield improved bounds over that obtained by adding all GMI cuts for most MIPLIB 3.0 problems.

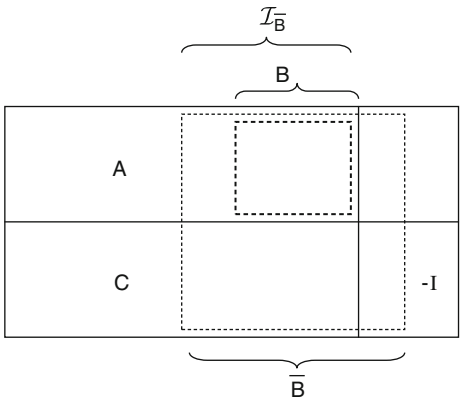

Fig. 1 The relationship between $\bar{B}$ and B

In Algorithm 1 we formalize the notion of a round of GMI cuts given a basis B and solution x^* of an LP relaxation of M. In a departure from Balas et al. [6], we do not require that x^* be basic with respect to B, nor that B be an optimal basis of the LP relaxation. If B is an optimal basis and x^* is basic with respect to B then Step 5 can be eliminated from Algorithm 1.

Algorithm 1: A round of GMI cuts

Input: A vector x^* satisfying $Ax^* = b$, a basis B of A, and a set of indices $S \subseteq \mathcal{I}$ identifying the integer variables.
Output: A list of GMI cuts.
1 **for** *each basic integer variable* x_j $(j \in S \cap \mathcal{I}_B)$ **do**
2 Compute the associated tableau row $x_j + \sum_{i \in \mathcal{I}_N} \bar{a}_i x_i = \bar{\beta}$.
3 **if** $\bar{\beta}$ *is not integral* **then**
4 Compute the corresponding GMI cut.
5 **if** *the GMI cut is violated by* x^* **then**
6 Store the cut in a list to be returned to the user.

Applying Algorithm 1 to an optimal basis B of L_1 and the associated basic solution x^*, and adding all the generated cuts to L_1, we obtain a linear program of the form

$$L_2 = \min\{cx : Ax = b,\ Cx \geq d,\ x \geq 0\},$$

where C and d represent the GMI cuts. If L_2 is written in equality form by adding surplus variables, then the cut-generation process can be repeated using an optimal basis and associated solution of L_2, as described in Algorithm 2. Clearly all GMI cuts obtained from L_1 are rank-1 cuts, but the GMI cuts obtained from L_2 may not have rank 1.

In this paper we explore new ways of generating rank-1 GMI cuts. We diverge from Algorithm 2 from $i = 2$ onwards. Instead of deriving GMI cuts from an optimal

Algorithm 2: A traditional GMI cutting plane algorithm

Input: An MIP with constraints $Ax = b, x \geq 0$ where integer variables have indices in $S \subseteq \mathcal{I}$. A number of iterations MAX_ITER.
Output: A set of cuts and a solution x^*.

1 $\bar{A} \leftarrow A,\ \bar{b} \leftarrow b,\ \bar{c} \leftarrow c,\ \bar{x} \leftarrow x.$
2 **for** $i = 1, \ldots, MAX\_ITER$ **do**
3 Solve $L_i := \min\{\bar{c}\bar{x} : \bar{A}\bar{x} = \bar{b}, \bar{x} \geq 0\}$ and obtain an optimal basis $\bar{B}$ and associated solution $\bar{x}^*$.
4 **if** $\bar{x}^*$ *is not integral* **then**
5 Use Algorithm 1 with input $\bar{x}^*$, $\bar{A}$, $\bar{b}$, $\bar{B}$ and S to obtain a list of GMI cuts.
6 Eliminate slack variables in the GMI cuts to get cuts in the x variables alone.
7 Update $\bar{A}$, $\bar{b}$, $\bar{c}$ and $\bar{x}$ to incorporate the new cuts and related slack variables.
8 **else**
9 Go to Step 10.
10 Eliminate slack variables and return $\bar{x}^*$ and cuts in original variables.

tableau of L_i for $i \geq 2$, we instead attempt to find a basis B of L_1 such that GMI cuts based on B are violated by the solution of L_i for $i \geq 2$. This procedure is described in Algorithm 3. Note that Step 7 of Algorithm 3 may not find violated cuts, whereas Step 5 of Algorithm 2 will always find cuts.

Algorithm 3: Our proposed GMI cutting plane algorithm

Input: An MIP with constraints $Ax = b, x \geq 0$ where integer variables have indices in $S \subseteq \mathcal{I}$. A number of iterations MAX_ITER.
Output: A set of rank-1 GMI cuts and a solution x^*.

1 $\bar{A} \leftarrow A,\ \bar{b} \leftarrow b,\ \bar{c} \leftarrow c,\ \bar{x} \leftarrow x.$
2 **for** $i = 1, \ldots, MAX\_ITER$ **do**
3 Solve $L_i := \min\{\bar{c}\bar{x} : \bar{A}\bar{x} = \bar{b}, \bar{x} \geq 0\}$ and obtain an optimal basis $\bar{B}$ and associated solution $\bar{x}^*$.
4 **if** $\bar{x}^*$ *is not integral* **then**
5 Use $\bar{x}^*$ as a "guide" to derive a basis B of L_1.
6 Project out slack variables from $\bar{x}^*$ to obtain x^* in the original space.
7 Use Algorithm 1 with input x^*, A, b, B and S to obtain a list of GMI cuts.
8 Update $\bar{A}$, $\bar{b}$, $\bar{c}$ and $\bar{x}$ to incorporate the new cuts and slack variables.
9 **if** $\bar{x}^*$ *is integral or Algorithm 1 failed to produce violated cuts* **then**
10 Go to Step 11.
11 Eliminate slack variables and return $\bar{x}^*$ and cuts in original variables.

The key step in Algorithm 3 is Step 5, where a new basis B of L_1 is chosen based on a solution $\bar{x}^*$ obtained from L_i using a two step-procedure. In the first step, we use $\bar{x}^*$ to select a subset of columns in A within which to search for a basis B. It is important that the selected subset of columns actually contains a basis of A. In the second step, we construct a basis from this subset. In Sect. 3 we describe a simple scheme to select a subset of columns in A, and focus on describing different ways of constructing a basis B from this subset. In Sect. 4 we describe a different way of performing the first step.

3 Alternate bases

In this section we are concerned with Step 5 of Algorithm 3. That is, given an optimal solution x^* of the linear program

$$L_i = \min\{cx : Ax = b,\ Cx \geq d,\ x \geq 0\} \tag{7}$$

for $i \geq 2$, we would like to obtain a basis B, not necessarily feasible, of the linear program,

$$L_1 = \min\{cx : Ax = b,\ x \geq 0\},$$

such that Algorithm 1 with input x^*, A, b, B, and S and returns some cuts violated by x^*. In the previous section, we described L_i as arising from L_1 via the addition of GMI cuts; thus, in that context, the rows of $Cx \geq d$ defined constraints satisfied by all integral solutions of L_1. In what follows, we will not use this fact. In Sect. 4, we will let $Cx \geq d$ stand for constraints which may not be satisfied by all integral solutions of L_1 (in particular, branching constraints).

It suffices to describe our procedure when $i = 2$ in (7). We first write L_2 in standard form. That is, if C has t rows, we add t surplus variables $s = (s_1, \ldots, s_t)$ and let

$$\bar{L}_2 = \min\{\bar{c}\bar{x} : \bar{A}\bar{x} = \bar{b},\ \bar{x} \geq 0\},\ \text{where}$$
$$\bar{A} = \begin{bmatrix} A & 0 \\ C & -I \end{bmatrix},\quad \bar{x} = \begin{pmatrix} x \\ s \end{pmatrix},\quad \bar{b} = \begin{pmatrix} b \\ d \end{pmatrix},\quad \bar{c} = (c, 0).$$

As A has full row-rank, so does $\bar{A}$.

Let $\bar{x}^* = (x^*, s^*)$ be a basic feasible solution of $\bar{L}_2$, and let $\bar{B}$ be the corresponding basis. Recall that $\mathcal{I} = \{1, \ldots, n\}$ is the index set of variables in L_1. The basis $\bar{B}$ consists of both variables in L_1, and surplus variables corresponding to the inequalities in $Cx \geq d$. Let $\mathcal{I}_{\bar{B}} \subseteq \mathcal{I}$ be the indices of the variables in L_1 which are basic with respect to $\bar{B}$. In our simplest implementation of the first step described at the end of the previous section, we search for a basis B of L_1 within $A_{\bar{B}}$, the columns in A with indices in $\mathcal{I}_{\bar{B}}$.

Clearly $\bar{B}$ has rank $m + t$ and has the form

$$\bar{B} = \begin{bmatrix} A_{\bar{B}} & 0 \\ C_{\bar{B}} & -I' \end{bmatrix},$$

where I' has t rows and is a submatrix of the $t \times t$ identity matrix. Therefore $A_{\bar{B}}$ has rank exactly m. Thus, the columns of $A_{\bar{B}}$ contain a basis B of A. We depict the relationships between the different bases above in Fig. 1. Any GMI cut based on B is valid for L_1, though such a cut need not be violated by x^*.

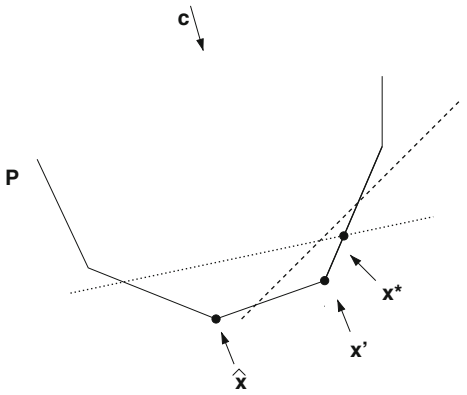

Fig. 2 The relationship between $\hat{x}$, x^* and x'

3.1 The feasible basis heuristic

Consider the solution x^* defined above, the set $\mathcal{I}_{\bar{B}}$, and the associated sub-matrix $A_{\bar{B}}$ of A. Let $x^*_{\bar{B}}$ and $c_{\bar{B}}$ denote the vectors obtained by taking the components of x^* and c contained in $\mathcal{I}_{\bar{B}}$, respectively. Clearly, $x^*_{\bar{B}}$ defines a feasible solution of $\min\{c_{\bar{B}} y : A_{\bar{B}} y = b, y \geq 0\}$, and therefore a basic, optimal solution of the above LP yields a basis B for L_1 such that $\mathcal{I}_B \subseteq \mathcal{I}_{\bar{B}}$. We define a heuristic to perform Step 5 of Algorithm 3 motivated by the above observation. Instead of deleting columns from A to obtain the above LP, we instead let

$$P' = \{cx : Ax = b, x \geq 0, x_i = 0 \,\forall i \notin \mathcal{I}_{\bar{B}}\}$$

and simply solve the problem $\min\{cx : x \in P'\}$ to obtain a basic optimal solution x' and corresponding basis B of L_1. Though there always exists an optimal basis with $\mathcal{I}_B \subseteq \mathcal{I}_{\bar{B}}$, a standard LP solver may in fact return a basis containing some of the variables which are fixed to zero, in the presence of degeneracy. Since this procedure is guaranteed to obtain a feasible basis, we call this procedure the Feasible Basis heuristic or FEAS. We also experimented with solving $\min\{0x : x \in P'\}$ to obtain a different feasible basis, but did not notice a significant difference in results.

In Fig. 2 we depict the optimal solution $\hat{x}$ of L_1 in the direction c, the point x^* obtained after adding a GMI cut (and solving L_2), and the point x' obtained by FEAS. In this figure, the dotted line refers to the first GMI cut, and the dashed line to a GMI cut obtained from B. Note that the point x^* lies on a facet of the polyhedron P, depicted by the thick line. Assume the surplus variable for the corresponding constraint is non-basic in L_2 (i.e., the corresponding column is not in $\bar{B}$). In P', we fix this surplus variable to 0; in other words, we only look for solutions to L_1 which lie on the face P'. The point x' in the figure is a basic optimal solution of $\min\{cx : x \in P'\}$. In this figure, x^* is violated by the second GMI cut.

If x^* is a basic solution of L_1, or equivalently, $|\mathcal{I}_{\bar{B}}| = m$, then $P' = \{x^*\}$. This suggests that if $\mathcal{I}_{\bar{B}}$ is not much larger than m, then extreme points in P' are somehow

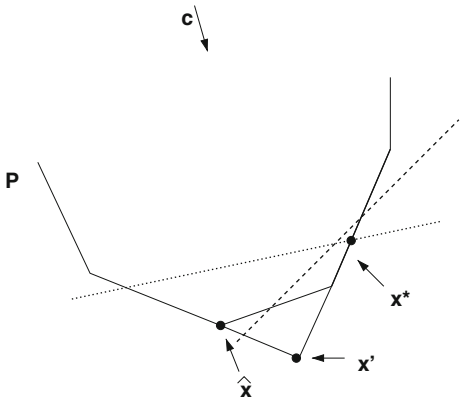

Fig. 3 A GMI cut from an infeasible solution x'

"similar" to x^*, and generating GMI cuts from such points might be a fruitful way of separating x^*. We make this notion more precise in Sect. 3.2.

Observe that an infeasible basis of P' may also yield a violated GMI cut. An example of this is depicted in Fig. 3, where the point x' is a basic infeasible solution, and yet the associated GMI cut (represented by the dashed line), is violated by x^*.

In Sect. 3.2, we explain why any basic solution of P' (feasible or infeasible) is likely to yield some violated GMI cuts when the constraint matrix A is very sparse, as is the case with many practical MIP instances.

3.2 Sparsity and degeneracy

Let x' be a basic (possibly infeasible) solution of $\min\{cx : x \in P'\}$, defined in the previous section, with associated basis B and non-basic columns N. Denote the ith tableau row for this basis as $(x_B)_i + \sum_{j \in \mathcal{I}_N} \bar{a}_{ij} x_j = \bar{\beta}_i$, where $(x_B)_i$ is an integer variable, $B^{-1}N = (\bar{a}_{ij})$, and $\bar{\beta} = B^{-1}b$. Assume $\bar{\beta}_i$ is non-integral and let $\mathcal{N}(i) = \{j \in \mathcal{I}_N : \bar{a}_{ij} \neq 0\}$. If $\mathcal{N}(i) \cap \mathcal{I}_{\bar{B}} = \emptyset$, then the GMI cut derived from this tableau row has *nonzero coefficients only for variables which have value 0 in x^* and is thus violated by x^**. This is because $(x_B)_i$ has a cut coefficient of zero, and thus the only nonzero cut coefficients correspond to $x_j (j \in \mathcal{I}_N \backslash \mathcal{I}_{\bar{B}})$.

Can such a tableau row exist? We argue that this can happen when $|\mathcal{I}_{\bar{B}} \backslash \mathcal{I}_B|$ is small relative to m, and when the constraint matrix A is very sparse. The second condition is often true for practical MIPs with many constraints and variables, including the MIPLIB problems. As for the first condition, for problems in MIPLIB 3.0, $|\mathcal{I}_{\bar{B}}|/|\mathcal{I}_B|$ is 1.1 on the average, and 1.05 for the MIPLIB 2003 problems, where $\bar{B}$ is the optimal basis of L_2, the LP relaxation after adding one round of GMI cuts.

Since x^* is feasible for L_1, it satisfies each tableau row for the basis B. Let a_k stand for the kth column of A. Then $Ax^* = b$ implies that

$$Bx_B^* + \sum_{k \in \mathcal{I}_{\bar{B}} \backslash \mathcal{I}_B} x_k^* a_k = b.$$

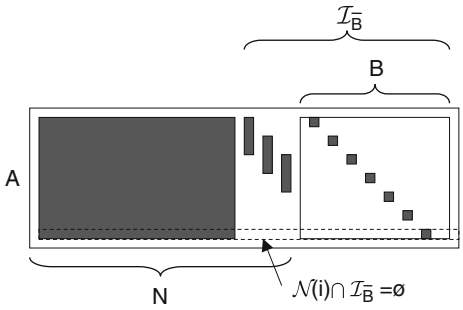

Fig. 4 An example of a sparse tableau

Let $\eta_k = B^{-1}a_k = (\bar{a}_{ik})_{i=1}^m$. Then the equation above and $Bx'_B = b$ imply that

$$x'_B - x^*_B = \sum_{k \in \mathcal{I}_{\bar{B}} \setminus \mathcal{I}_B} x^*_k \eta_k. \tag{8}$$

In other words, x' differs from x^* in the ith component of $\mathcal{I}_B$ if and only if $\sum_{k \in \mathcal{I}_{\bar{B}} \setminus \mathcal{I}_B} x^*_k \bar{a}_{ik}$ is nonzero.

Observe that $|\mathcal{I}_{\bar{B}} \setminus \mathcal{I}_B| \leq t$. If the number of nonzeros in each vector η_k with $k \in \mathcal{I}_{\bar{B}} \setminus \mathcal{I}_B$ is less than m/t, then there are fewer than m nonzeros in the tableau columns with indices in $\mathcal{I}_{\bar{B}} \setminus \mathcal{I}_B$. This implies that some tableau rows satisfy the condition $\mathcal{N}(i) \cap \mathcal{I}_{\bar{B}} = \emptyset$; See Fig. 4 for an illustration of this condition. If these rows correspond to basic integral variables with non-integral values in $\bar{\beta}$, then there exist GMI cuts based on B which are violated by x^*. This analysis did not assume that B is feasible, just that $B^{-1}a_k$ is sparse, for $k \in \mathcal{I}_{\bar{B}} \setminus \mathcal{I}_B$.

Even if the columns $B^{-1}a_k$, with $k \in \mathcal{I}_{\bar{B}} \setminus \mathcal{I}_B$ are not too sparse, a GMI cut based on the ith tableau row may be violated if $x^*_k = 0$ for many $k \in \mathcal{I}_{\bar{B}} \setminus \mathcal{I}_B$ (as x^* is a basic solution of L_2, the above condition can hold only in the presence of degeneracy). More generally, as $(x' - x^*)_j = 0$ if $j \notin \mathcal{I}_{\bar{B}}$, (8) implies that if $\max_{k \in \mathcal{I}_{\bar{B}} \setminus \mathcal{I}_B} \{x^*_k\}$ is small, then so is $||x' - x^*||_\infty$ and a GMI cut based on the ith tableau row violated by x' is likely to be violated by x^*. These arguments suggest the following two ideas to find a suitable B.

1. Find B such that $B^{-1}a_k$ is sparse for $k \in \mathcal{I}_{\bar{B}} \setminus \mathcal{I}_B$.
2. Find B such that $\sum_{k \in \mathcal{I}_{\bar{B}} \setminus \mathcal{I}_B} x^*_k$ is small.

3.3 The sparse basis heuristic

As suggested above, we next consider a heuristic to find an $m \times m$ basis B of a rectangular matrix $\tilde{A} \in \mathbb{R}^{m \times k}$ for some $k > m$ such that $B^{-1}N$ is sparse. More precisely, we let $\tilde{A} = A_{\bar{B}}$. One approach to this problem is to solve the related problem: find a basis B such that B^{-1} is sparse. This latter problem has recently been studied in [35]. If B^{-1} is sparse, than $B^{-1}N$ is sparse assuming $\tilde{A}$ (or N) is sparse.

We use a heuristic different from the one proposed in [35], where the authors use bipartite matchings in auxiliary graphs to find an appropriate B. We consider the sparse

LU factorization approach of Suhl and Suhl [36] which is based on a combination of the Markowitz criterion [33] for reducing fill-in during pivoting and threshold pivoting for numerical stability. In particular, we consider the LU implementation in QSopt 1.0 [3] written by David Applegate and based on the Suhl and Suhl paper. We modify the code so that it takes as input a rectangular matrix (namely A) instead of a square matrix and returns a factorization of $\tilde{A}$ as $\tilde{A} = LU$, where L is a square, lower triangular matrix with ones on the diagonal, and U is rectangular and upper triangular. That is, $U = (U'U'')$, where U' is a non-singular upper triangular matrix, and U'' is a rectangular $m \times (k - m)$ matrix. The Markowitz criterion helps in keeping U sparse. Finally we define $B = LU'$ and therefore

$$B^{-1}N = (U')^{-1}L^{-1}N = (U')^{-1}U''.$$

Our hope is that as U' and U'' are sparse $(U')^{-1}U''$ will be sparse. In general, the basis B generated by this approach need not be a feasible basis. We refer to this heuristic as SPARSE.

When we apply SPARSE to L_2 obtained after the first round of GMI cuts, the basis found has the property that $\mathcal{N}(i) \cap \mathcal{I}_{\bar{B}} = \emptyset$ for some tableau row in 31 out of 54 MIPLIB 3.0 problems in our test set, and 17 out of 20 problems in MIPLIB 2003.

3.4 The greedy heuristic

Given non-negative weights on the columns of a full row-rank matrix $A \in \mathbb{R}^{m \times n}$, in polynomial time one can obtain a maximum weight basis B of A via the greedy algorithm for matroids [23]: start off with an empty set of columns $\mathcal{B}$, sort the columns of A by decreasing weight, and iterate through the sorted columns of A adding them to $\mathcal{B}$ if they are not linearly dependent on the columns already in $\mathcal{B}$. At the end of this iterative process, $\mathcal{B}$ will have m columns and form a basis of A.

This motivates a heuristic GREEDY which assigns large weights to columns of A which we want to be present in the basis, and small weights to the remaining columns of A. We sort the variables by decreasing value of x_i^* for $i \in \mathcal{I}_{\bar{B}}$ (recall that all variables are assumed to have a lower bound of 0, and no upper bounds). We then use the greedy algorithm to select a maximum weight basis B'. We thus attempt to choose B such that $\sum_{k \in \mathcal{I}_{\bar{B}} \setminus \mathcal{I}_B} x_k^*$ is as small as possible, precisely the criteria suggested at the end of Sect. 3.2.

For practical MIPs which have inequality constraints, and need slacks to be expressed in standard form, we distinguish between the slack and structural variables in the original constraints. In order to get a sparse basis, we assign a weight of ωx_i^* for some large $\omega > 1$ if x_i^* is a slack variable. One drawback of this idea is that it may lead to too many continuous (slack) variables in the basis B, leaving few integral variables in B, and thus allowing the generation of only a small number of GMI cuts based on B.

3.5 The random basis heuristic

It is possible that the bases returned by the heuristics FEAS, SPARSE, and GREEDY do not yield GMI cuts violated by x^*, in which case Algorithm 3 terminates. To let

Algorithm 3 proceed when the above heuristics fail, and also to test if these heuristics truly yield useful bases (yielding violated GMI cuts) of A with columns in $\mathcal{I}_{\bar{B}}$, we are interested in generating a random basis of A, i.e., sampling from a uniform distribution over all bases of A. This seems to be a hard problem, and a polynomial-time method to generate a random basis of A seems not to be known.

We instead implement a simple randomized algorithm. We assign weights drawn uniformly from [0, 1], and then apply the greedy algorithm discussed in the previous section. This algorithm does not generate a basis uniformly at random: consider the matrix

$$A = \begin{bmatrix} 1 & 2 & 1 & 1 \\ 0 & 0 & 1 & 2 \end{bmatrix}$$

with possible bases

$$B_1 = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}, \quad B_2 = \begin{bmatrix} 2 & 1 \\ 0 & 1 \end{bmatrix}, \quad B_3 = \begin{bmatrix} 1 & 1 \\ 0 & 2 \end{bmatrix}, \quad B_4 = \begin{bmatrix} 2 & 1 \\ 0 & 2 \end{bmatrix}, \quad B_5 = \begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix}.$$

Assigning random weights from [0, 1] to the columns of A implies that the greedy algorithm iterates through each permutation of the columns of A with equal probability. However, there are 24 such permutations, but 5 possible bases, and thus not all bases are equally likely to be generated by the greedy algorithm. In particular, B_5 will be chosen with probability $4/24$ while each of the remaining bases will be chosen with probability $5/24$. The above heuristic is easy to implement and is also quite useful as we discuss in Sect. 5. We repeat the generation process (up to a default limit of 5 times) till a basis yields violated cuts.

4 Branching

Recall that the constraints $Cx \geq d$ in L_2 were not required to be valid for M in order for the GMI heuristics described in Sect. 3 to work. In particular if the constraints in $Cx \geq d$ include branching constraints obtained at nodes of a branch-and-bound tree, the heuristics described earlier can still be used to obtain globally valid rank-1 GMI cuts from each node. In fact, assume $\tilde{x}$ is a basic solution of the LP relaxation at a branch-and-bound node. Let $Cx \geq d$ represent the constraints which are only locally valid at that node of the tree (this will include the branching constraints), and let $Ax = b, x \geq 0$ be the original constraints. Using Algorithm 1, with input $\tilde{x}$ and an appropriate basis of L_1, we can generate cuts which are globally valid and may separate $\tilde{x}$.

Branching can also be used in a very different way. Consider system L_2, an optimal basis $\bar{B}$ and its corresponding solution x^*. In Sect. 3 we presented a number of heuristics which seek to obtain a basis B of L_1 contained in the columns with indices in $\mathcal{I}_{\bar{B}}$. Once $\mathcal{I}_{\bar{B}}$ is determined, these heuristics did not use $\bar{B}$ or x^* in generating GMI cuts. Thus, the heuristics presented could be made to work on any set $\mathcal{I}' \subseteq \mathcal{I}$ such that the corresponding columns contain a basis of L_1. In this section we deviate from the previous procedure by considering such sets $\mathcal{I}'$ different from $\mathcal{I}_{\bar{B}}$, and then using the heuristics described earlier to find a basis.

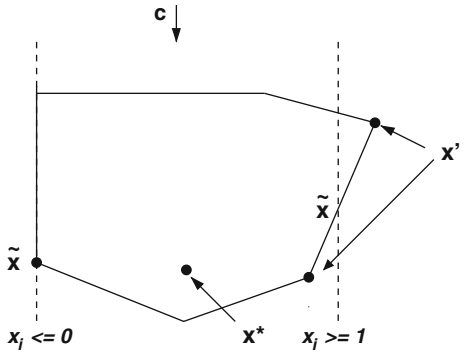

Fig. 5 The branch and gather heuristic

Let M_2 be the MIP obtained by adding back the integrality constraints of L_2. That is,

$$M_2 = \min\{cx : Ax = b,\ Cx \geq d,\ x \geq 0,\ x_i \in \mathbb{Z}\ \forall i \in S\}. \tag{9}$$

We explore the branch-and-bound tree of M_2 for a limited number of nodes. For this, we solve the LP relaxation at each node and branch as usual, always exploring the node with the minimum bound (i.e, best-bound branching). At each node of the search tree, we let $Cx \geq d$ correspond to the rank-1 GMI cuts in (9) and *the branching constraints defining the node*. Let the node solution be $\tilde{x}$ with associated basis $\tilde{B}$. Then the heuristics described in Sect. 3 will search for a basis B of L_1 contained in the columns in $\mathcal{I}_{\tilde{B}}$ $(= \mathcal{I}')$, but will check if the corresponding GMI cuts separate x^*, the root solution, not $\tilde{x}$. In Fig. 5, x^* is contained in the interior of the LP relaxation of a MIP. As x^* does not lie on a face of the LP relaxation, our previous heuristics will not find a basis as there is no basis of L_1 contained in $\mathcal{I}_{\tilde{B}}$. However, if we consider the nodes defined by $x_i \leq 0$ and $x_i \geq 1$ with the node solutions depicted by $\tilde{x}$, we see that there are vertices of the LP relaxation (denoted by x') lying on the same faces as $\tilde{x}$ (for the branch $x_i \leq 0$, $\tilde{x} = x'$) and thus our previous heuristics will find some basis contained in $\mathcal{I}_{\tilde{B}}$.

The intuition behind exploring the node with the minimum bound is that we will explore the node "most similar" to the root. Since this algorithm works by exploring the branch-and-bound tree in order to gather cuts for the original root problem, we call this the BRANCH AND GATHER heuristic, abbreviated as BG. When using this algorithm we can use any subset of the heuristics described in the previous section in order to find bases contained in the columns in $\mathcal{I}_{\tilde{B}}$. In our default implementation, we use FEAS and GREEDY and branch for five nodes.

5 Computational results

In this section we present lower bounds for problem instances in MIPLIB 3.0 [12] and MIPLIB 2003 [1] obtained by running our code with different basis generation heuristics with a time limit of one hour. In the case of MIPLIB 3.0, we ignore instances which do not have any integrality gap, namely *dsbmip, enigma* and *noswot*, and instances

where one round of GMI cuts close 100% of the integrality gap, namely *air03*, *10teams*, and *mod010*. We also omit instances for which Dash et al. [22] and Balas and Saxena [8] report that optimizing (approximately) over the MIR closure does not change the bound from the LP relaxation value. These are *markshare1*, *markshare2*, *pk1*, *stein27* and *stein45*. In other words, we omit instances where there is no scope for improvement over and above one round of GMI cuts and are left with 54 instances out of 65 from MIPLIB 3.0.

There are some instances in MIPLIB 2003 for which no integer solution is known, namely *liu*, *momentum3*, *stp3d*, and *t1717*, and we omit them. We also omit *disctom*, as it has no integrality gap, and *manna81*, where GMI cuts close 100% of the integrality gap. We are left with 20 instances from MIPLIB 2003 not contained in MIPLIB 3.0.

5.1 Implementation details

After adding slack variables, we can assume MIP instances have the form

$$\min\{cx : x \in P, x_i \in \mathbb{Z}\ \forall i \in S\},$$

where $P = \{x \in \mathbb{R}^n : Ax = b,\ l \leq x \leq u\}$ for some $l, u \in \mathbb{R}^n \cup \{-\infty, +\infty\}$, and $S \subseteq \{1, \ldots, n\}$. Given a tableau row based on any basis of the LP relaxation of a MIP, and upper and lower bounds on variables, we generate GMI cuts using the CglTwomir cut generator of COIN-OR [17] (which is available, for example, as a part of CBC 2.2.0) with the numerical parameters and options described in [21], except for one change, described in the next paragraph. The GMI cut generated varies with the point x^* being separated if x^* is not basic with respect to the basis defining the tableau row.

In general, when given a tableau row based on a basic optimal solution of the LP relaxation, one usually computes a GMI cut if the corresponding basic variable is of integer type and if the right-hand-side is fractional. This is guaranteed to yield a violated cut as the corresponding non-basic variables have zero value. In our application, we try to separate a solution x^* which is basic with respect to $\bar{B}$, but is non-basic with respect to B, where $\mathcal{I}_B \subset \mathcal{I}_{\bar{B}}$. As in Sect. 3, B stands for a basis of the original LP relaxation, whereas $\bar{B}$ stands for a basis of this relaxation plus previously generated rank-1 GMI cuts. Therefore we generate GMI cuts based on B even when a basic integer variable has integral value.

In our default separation algorithm, denoted as DEF, the first heuristic invoked is FEAS. If it does not find any violated cuts then SPARSE is invoked, then GREEDY, then RANDOM up to five times, and finally BG with a limit of five nodes.

5.2 Bounds at the root node

In Tables 1 and 2, we compare our results on MIPLIB 3.0 problems obtained by the algorithm DEF with those obtained in [22] and [8] by solving MIP models to find violated MIR cuts. To make the comparison with [22] as fair as possible, we run our code using the same hardware and software settings, i.e., we use the same machine,

Table 1 IPs of MIPLIB 3.0

Instance	1GMI	# Cuts DEF	% Gap closed	Time DEF	# Cuts MIR	% Gap closed	Time MIR	% Gap closed	Time split
air04	8.08	7,992	19.37	3,745.26	294	9.18	3,600	91.23	864,360
air05	4.65	6,275	12.92	3,601.46	246	12.38	3,600	61.98	24,156
cap6000	41.65	34	62.41	14.65	316	49.77	3,600	65.17	1,260
fast0507	1.66	2,030	1.71	4,257.80	318	1.66	3,600	19.08	304,331
gt2	76.50	99	96.61	0.29	256	98.38	2,618	98.37	599
harp2	22.43	1,136	67.31	240.81	523	58.48	108	46.98	7,671
l152lav	1.55	2,362	48.65	402.71	128	6.41	3,600	95.20	496,652
lseu	7.74	61	84.63	0.12	350	91.84	3,600	93.75	32,281
mitre	82.86	1,031	100.00	59.51	1,126	100.00	1,396	100.00	5,330
mod008	20.89	90	62.50	0.62	203	98.95	201	99.98	85
nw04	66.08	192	100.00	84.74	270	93.30	3,600	100.00	996
p0033	56.82	53	83.18	0.10	110	87.42	2,552	87.42	429
p0201	16.89	773	61.28	6.66	990	74.31	3,600	74.93	31,595
p0282	3.70	431	96.85	3.55	1,419	99.55	3,600	99.99	58,052
p0548	41.04	501	94.26	2.67	1,317	96.11	3,600	99.42	9,968
p2756	0.46	466	96.85	17.05	671	57.57	3,600	99.90	12,673
seymour	8.35	12,551	20.79	3,613.54	559	8.35	3,600	61.52	775,116

operating system, compiler (g++ version 4.1.3 on a 1,452 MHz powerpc running AIX 5.1), and the same version of ILOG CPLEX (release 9.1). The results in [8] were obtained on a different machine type and operating system, but with a similar version of CPLEX (release 9.0) and around the same timeframe. In all other tables, we use a faster machine (a 4,208 MHz powerpc) and ILOG CPLEX 11.2.

Table 1 contains results for the pure integer problems in MIPLIB 3.0, and Table 2 contains results for the mixed-integer problems. In column 2 we give the percentage integrality gap closed by one round of GMI cuts, i.e., cuts from the optimal tableau of L_1 (henceforth referred to as 1GMI). In columns 3, 4 and 5, we give, respectively, the number of generated cuts, percentage integrality gap closed, and running time (in seconds; it includes LP resolve time after adding cuts), respectively, for DEF, and in columns 5, 6 and 7, we give the same information for the code in [22]. In columns 8 and 9 we give, respectively, the percentage integrality gap closed in [8] and the time to do so. The numbers in columns 4 and 5 indicate that for many problems, especially the smaller ones, our heuristic often obtains bounds comparable to those in [22] in much less time. For example, for *cap6000, gt2, p0282, fiber* and *pp08a*, DEF closes a significant fraction of the remaining integrality gap after the first round of GMI cuts, and obtains a comparable bound to those in [22] and [8] in a fiftieth of the time or less. Further, we improve on the best-known bounds for the objective function value over the MIR closure for *mkc*, *harp2* and *rentacar*.

A point to note is that as our heuristics primarily involve solving LPs, our code is not affected very much by the quality of the components of the IP solver we use related

Table 2 MIPs of MIPLIB 3.0

Instance	1GMI	# Cuts DEF	% Gap closed	Time DEF	# Cuts MIR	% Gap closed	Time MIR	% Gap closed	Time split
arki001	29.26	293	35.47	394.84	133	33.94	3,600	83.05	193,536
bell3a	60.15	130	74.03	0.82	404	99.60	3,600	65.35	102
bell5	14.53	49	23.65	0.32	629	92.95	3,600	91.03	2,233
blend2	16.36	28	21.43	2.07	2,815	30.63	3,600	46.52	552
dano3mip	0.10	1,104	0.28	3,688.43	124	0.10	3,600	0.22	73,835
danoint	1.73	1,942	1.73	235.69	1,044	1.73	3,600	8.20	147,427
dcmulti	45.34	446	91.37	12.40	3,866	97.81	3,600	100.00	2,154
egout	55.93	85	98.67	0.30	264	100.00	10	100.00	18,179
fiber	64.26	551	98.12	14.78	329	94.70	3,600	99.68	163,802
fixnet6	10.87	602	86.58	15.55	4,766	93.38	3,600	99.75	19,577
flugpl	11.74	10	11.74	0.01	28	80.23	3,600	100.00	26
gen	60.23	190	91.81	10.50	115	100.00	825	100.00	46
gesa2	27.22	292	85.61	45.46	1,378	99.70	3,600	99.02	22,808
gesa2_o	30.12	345	58.84	78.47	1,640	96.05	3,600	99.97	8,861
gesa3	45.75	850	92.59	119.48	892	74.83	3,600	95.81	30,591
gesa3_o	49.11	870	93.39	108.54	1,382	70.82	3,600	95.20	6,530
khb05250	74.91	141	99.46	3.49	555	100.00	146	100.00	33
mas74	6.67	81	8.46	0.63	12	6.68	0	14.02	1,661
mas76	6.42	117	10.34	1.13	11	6.45	0	26.52	4,172
misc03	5.86	571	20.56	6.15	992	37.71	3,600	51.70	18,359
misc06	10.73	69	86.15	34.46	2,074	99.84	792	100.00	229
misc07	0.72	975	2.12	11.08	1,678	11.25	3,600	20.11	41,453
mkc	5.18	21,149	49.30	2,058.05	4,259	13.42	3,600	36.16	51,519
mod011	17.11	2,685	32.19	3,628.13	1,673	17.41	3,600	72.44	86,385
modglob	17.09	257	75.30	8.59	7,060	80.04	1,677	92.18	1,594
pp08a	52.36	263	93.75	3.12	1,687	95.76	3,600	97.03	12,482
pp08aCUTS	30.94	347	83.27	21.52	2,126	88.74	3,600	95.81	5,666
qiu	1.76	4,690	25.89	3,715.85	2,142	29.19	3,600	77.51	200,354
qnet1	14.27	943	79.31	437.03	784	66.22	3,600	100.00	21,498
qnet1_o	26.62	698	93.47	196.73	587	83.78	3,600	100.00	5,312
rentacar	29.05	308	44.95	1,123.07	265	23.40	3,600	0.00	0
rgn	4.71	176	99.59	0.54	1,142	99.60	3,600	100.00	222
rout	0.32	1,008	35.48	180.27	9,393	22.60	3,600	70.70	464,634
set1ch	38.11	673	83.59	30.66	694	76.47	3,600	89.74	10,768
swath	17.12	843	33.96	64.99	1,476	33.93	3,600	28.51	2,420
vpm1	9.09	255	89.78	5.10	386	96.30	387	100.00	5,010
vpm2	12.48	295	61.25	8.03	427	77.71	243	81.05	6,012

Table 3 Performance on MIPLIB 2003 problems

Instance	1GMI	# cuts DEF	% Gap closed	Time DEF
a1c1s1	18.41	1,898	54.32	3,578.97
aflow30a	11.88	516	46.82	98.09
aflow40b	5.33	741	33.47	1,163.44
atlanta-ip	1.08	8,490	1.08	3,853.58
glass4	0.00	590	0.00	1.51
momentum1	38.17	1,867	41.14	4,169.05
momentum2	40.65	4,234	40.66	3,845.24
msc98-ip	44.58	48,073	46.79	3,601.84
mzzv11	11.43	19,069	24.70	3,701.73
mzzv42z	12.01	14,133	51.83	4,199.74
net12	6.89	14,773	13.58	3,631.19
nsrand-ipx	36.24	766	80.47	1,545.22
opt1217	19.12	22,944	32.28	3,601.71
protfold	5.04	13,145	11.70	3,648.65
rd-rplusc-21	0.00	2,558	0.00	1,491.32
roll3000	7.03	2,293	51.94	1,880.39
sp97ar	7.60	2,239	28.43	3,649.03
timtab1	23.59	1,269	77.80	21.62
timtab2	18.02	2,269	69.18	47.65
tr12-30	60.27	1,170	87.31	42.15

to cuts, branching etc., but is mainly affected by the LP solver component. Thus, on average, the bounds we obtain in Tables 1 and 2 are not very different from the bounds we obtain with CPLEX 11.2 in Table 4.

Table 3 contains the percentage gap closed with DEF for problems in MIPLIB 2003, and the columns have the same meaning as the first four columns in Tables 1 and 2. For these instances, the initial LP relaxation and the subsequent ones obtained after adding cuts are quite a bit harder to solve than in the case of MIPLIB 3.0 instances. Note that for 11 out of 20 problems, the time limit is reached. For a few instances, the time limit is reached before FEAS stops generating cuts, and we do not even invoke any other heuristics.

In Table 4, we report on the average integrality gap closed with different separation algorithms and combinations of heuristics. As in earlier tables, "1GMI" stands for one round of GMI cuts. We also give the average gaps closed in Dash et al. [22] (indicated by "DGL") and [8] (indicated by "Balas–Saxena"). These two papers do not contain results for MIPLIB 2003. We give the average gap closed by the `CglLandP` cut generator [9] in COIN-OR's CBC 2.2.0 for MIPLIB 3.0 problems. To obtain this number, we invoke the generator for one round (as opposed to 10 rounds in [4]), so that only rank-1 GMI cuts are generated. We use the default settings except that we do not limit the number of cuts generated. We could not obtain the average gap closed for MIPLIB 2003 problems owing to numerical problems in the COIN-OR LP Solver in some cases. We do not give the numbers of Bonami and Minoux [10], who close a

Table 4 Average performance on MIPLIB problems

	MIPLIB 3.0	MIPLIB 2003
1GMI	26.09	18.37
DGL	62.53	–
Balas-Saxena	76.52	–
CglLandP	30.21	–
FEAS	43.96	27.64
SPARSE	38.56	29.25
GREEDY	42.62	31.03
RANDOM	41.67	25.78
RANDOM5	48.10	29.75
ALL	52.39	35.51
ALL-FEAS	48.51	35.49
ALL-SPARSE	50.52	34.10
ALL-GREEDY	51.90	33.38
ALL-RANDOM	51.13	35.05
DEF	62.16	39.68
DEF+BG100	64.58	40.82

non-trivial fraction of the integrality gap with their heuristic to find rank-1 GMI cuts, as their code works only for 0–1 problems.

We then report on each of the heuristics used in isolation. For the problems in MIPLIB 3.0, FEAS seems to be the best, while SPARSE seems to be the weakest heuristic, though by a small margin. However, this ordering of the heuristics does not hold for the MIPLIB 2003 problems, where GREEDY is the best and RANDOM is the worst performing. We suspect that for the small instances in MIPLIB 3.0, any basis in $\mathcal{I}_{\bar{B}}$ yields useful cuts, but that is not the case for the larger instances in MIPLIB 2003. In the latter case, the heuristics which use the ideas in Sect. 3.2, namely SPARSE and GREEDY, are better than the other two heuristics.

The hybrid heuristic ALL, stands for all the basic heuristics (not BG) executed in the same order as in DEF, though RANDOM is executed only once. Turning off the different heuristics in ALL, yields bounds somewhat consistent with the ranking of the different basic heuristics; turning off the best heuristic leads to the biggest drop in the lower bounds. The heuristics seem to be better by a significant margin than any individual one when combined together. Further, running RANDOM up to five times whenever a point needs to be separated yields a fairly good bound which is better than any of the heuristics individually (for MIPLIB 3.0), but is weaker than ALL, by a nontrivial margin. We therefore claim that we can get a better bound by combining our heuristics than we can get by simply generating the same number of random bases at each iteration. The row indicated by DEF gives the performance of the default heuristic DEF (it includes BG for 5 nodes and RANDOM up to 5 times). Finally, "DEF+BG100" stands for the bounds obtained when we allow BG in DEF to generate up to 100 nodes per invocation (and use up a lot of computing time).

Table 5 Detailed statistics on all MIPLIB problems

	One round			Many rounds		
	Bound	# Cuts	Density	# Bases	Time/round	Time/lp
MIPLIB 3.0						
1GMI	26.09	1	0	1	1	1
FEAS	31.23	0.80	2.49	34.20	1.15	3.12
SPARSE	30.84	0.55	2.86	10.00	1.02	0.74
GREEDY	32.40	0.67	7.93	26.83	5.20	4.45
MIPLIB 2003						
1GMI	18.37	1	0	1	1	1
FEAS	20.78	0.79	2.75	24.11*	1.46*	6.43*
SPARSE	21.65	0.55	2.52	17.60	1.35	3.85
GREEDY	21.17	0.69	3.14	31.25	2.82	7.40

In Table 5, we give some performance details of our individual heuristics (other than BG, which is a hybrid). In columns 2–4, we report on the effect of a single round of cutting-plane generation after adding GMI cuts from the initial tableau. In other words, we are generating cuts to separate x^*, the basic optimal solution of L_2. In column 2, we give the average gap closed. Note that just one round of any of our heuristics yields a non-trivial improvement over 1GMI. In column 3, we give the average of the ratio of the number of violated cuts generated by a heuristic to the number of violated cuts from 1GMI. For example, on the average, for MIPLIB 3.0 problems, FEAS generates about 80% the number of violated cuts as compared to 1GMI. In column 4, we give a measure of taleau density. This is the average number of nonzero entries corresponding to nonzero variables in $\mathcal{I}_{\bar{B}}$ in a tableau row associated with a basic integer variable minus one (for the basic variable corresponding to the row). This number is zero for the tableau rows returned by 1GMI.

The remaining columns give results from running the heuristics for multiple rounds subject to a time limit of one hour. In columns 5–7, we give, respectively, the number of generated bases (= rounds) and measures of average time to generate a basis, and average time to resolve the LP after adding the generated cuts. The number in column 6 is computed as follows: for each problem instance, we compute the average time per round to generate cuts (this consists of the time to find a basis, compute the associated tableau, and generate the violated cuts) normalized by the time to generate the first round of GMI cuts (computing the tableau + cut generation). We then average this number across all problems in MIPLIB 3.0 and MIPLIB 2003. The number in the last column is computed similarly, by taking the ratio of the average LP resolve time to the LP resolve time after adding GMI cuts for a problem, and then averaging this ratio.

On the average, each round of FEAS takes just 15% more time than 1GMI, whereas a round of SPARSE takes just 2% more time. Our implementation of GREEDY is more expensive. The point of this column is to demonstrate that the time for an invocation of FEAS or SPARSE is comparable to the time to generate the first round of GMI cuts, and not too much more with GREEDY. In contrast, a round of cutting-plane generation via solving an auxiliary MIP, as in [22] and [8], can be significantly more expensive. Finally, the average LP resolve time is within a factor of 5 for all heuristics.

In our implementation, we do not delete cuts, and hence LP resolves in later rounds are more expensive per added cut than in earlier rounds. On the other hand, we usually find and add fewer violated cuts in later rounds, and thus the ratio can be less than one.

One can interpret columns 5–7 as follows: on the average FEAS is 4.23 (=1.15+3.12) times as time-consuming per round as 1GMI, and runs for 34.20 rounds; thus FEAS consumes about $144.67 = 34.20 \times 4.23$ times the running time of 1GMIto improve the integrality gap closed from 26.09 to 43.96% (see Table 4). On the other hand, SPARSE improves the gap closed to 38.56%, but is only about 18 times as expensive as 1GMI. In the last three columns for FEAS with MIPLIB 2003 problems (marked by a '*'), we omit the instance *opt1217* as it skews these numbers significantly: for this instance FEAS generates 1,803 bases.

5.3 Branch-and-cut

We now measure the effect of adding our cuts as globally valid cuts in a branch-and-cut tree for some instances. Though a branch-and-cut algorithm often generates fewer nodes than a pure branch-and-bound algorithm (with the same branching rules) in achieving a given lower bound on the optimal solution, it often takes more time to do so. The extra time spent in cut generation and solving harder LPs is often counter-productive. This effect is especially pronounced for the small and easy problems in MIPLIB 3.0, many of which can be solved in a few hundred nodes and a fraction of a second. For example, CPLEX 11.2 solves *p0201* in 107 nodes and 0.14 s and *p0282* in 427 nodes and 0.09 s. Our default algorithm DEF takes over 10 s just to generate rank-1 GMI cuts at the root node for *p0201* (this is with CPLEX 11.2, and not with CPLEX 9.1 as in Table 1).

Thus our moderately expensive cutting plane algorithm is unlikely to be effective except where many thousands of nodes are needed to solve a MIP to optimality. We therefore only consider problems from MIPLIB 2003 not contained in MIPLIB 3.0 (these are harder in general), and focus on a few problems where our cut generation heuristics are not too time-consuming, i.e., DEF terminates in less than an hour in Table 3 and yet improves the lower bound compared to the first round of GMI cuts. These problems are *aflow30a*, *aflow40b*, *nsrand-ipx*, *roll3000*, *timtab1*, *timtab2* and *tr12-30*. We also consider *a1c1s1* for the first experiment discussed below; if FEAS alone is used to generate cuts, it yields a significantly better bound than one round of GMI cuts, yet terminates in less than an hour. We perform three different experiments to measure the effectiveness of our cuts.

In our first experiment, we check if there are any problems from the list above for which branch-and-cut (with our cuts) is preferable to branch-and-bound. To do this, we apply Algorithm 3 with FEAS until we do not find any violated cuts (MAX_ITER is set to infinity). Starting from the augmented problem, we then perform branch-and-bound for 30 min and compare its behaviour with branch-and-cut for 30 min where we add cuts generated by FEAS at nodes of the branch-and-cut tree. We use CPLEX function calls to execute branch-and-bound via its default branching strategy. We execute branch-and-cut with the same settings, except that we use CPLEX cut callbacks to invoke FEAS. In both cases, we turn off all CPLEX cutting planes, presolve routines,

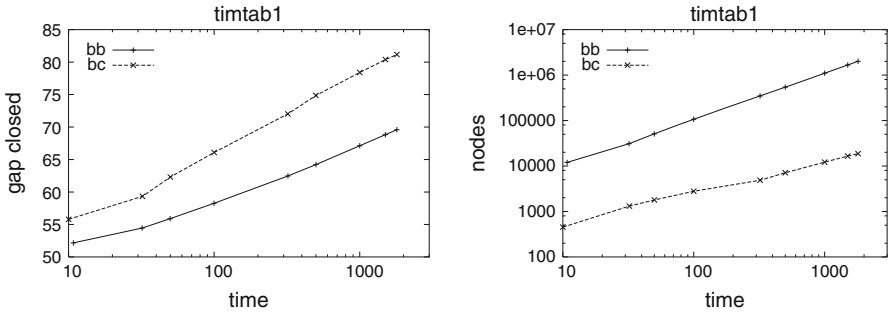

Fig. 6 Performance on timtab1

and the dynamic search routine. The reason for the last choice is that CPLEX automatically turns off dynamic search during banch-and-cut when user-defined cuts are added. Note that CPLEX may execute our cut generation heuristic even at the root node (say after fixing variables based on reduced costs).

The specific metrics of comparison are the time and number of nodes required to close a given integrality gap. The purpose of this experiment is twofold. Firstly, we want to verify that we can generate globally valid cuts at nodes of a branch-and-cut tree which reduce the number of nodes needed to attain a given lower bound. Secondly, we want to check if our cuts help to increase the lower bound faster than pure branch-and-bound for any problems in MIPLIB 2003. For five out of the eight problems we considered, the answer to the second question is true, and is false for the *aflow* problems, and for *roll3000*.

In Fig. 6a, we plot the integrality gap closed as a function of time for both branch-and-cut (denoted by 'bc') and branch-and-bound (denoted by 'bb') on *timtab1*, whereas in Fig. 6b we plot the number of nodes explored as a function of time. We use a logarithmic scale for time, and for the number of nodes. The time instants at which we measure the nodes and gap closed are approximately equal for the 'bb' and 'bc' curves, and is the same across the two figures. Clearly the lower bound improves at a faster rate for branch-and-cut than for branch-and-bound on *timtab1*, while the number of nodes increases at a slower rate, and is more than a hundred times smaller at the end of 30 min. Therefore our cuts clearly help in reducing the search tree size and the time required to obtain a given lower bound for *timtab1*. A similar behaviour can be observed for *timtab2*.

On the other hand, for *roll3000* (see Fig. 7) by the time one node has been fully explored by branch-and-cut at the end of about 50 s with a resulting integrality gap of about 30%, branch-and-bound explores almost a 1,000 nodes and closes about 40% of the integrality gap. Subsequently, the rate of growth of nodes in branch-and-bound is slightly smaller than the rate of growth of nodes in branch-and-cut (possibly because of the better bound). Thus, at the end of 30 min, a significantly better lower bound is obtained via branch-and-bound even though about 50,000 nodes are explored as opposed to 561 nodes in the case of branch-and-cut. Clearly, our cut generation process is too expensive for this problem. A similar behaviour can be observed in the case of *aflow30a* and *aflow40b*.

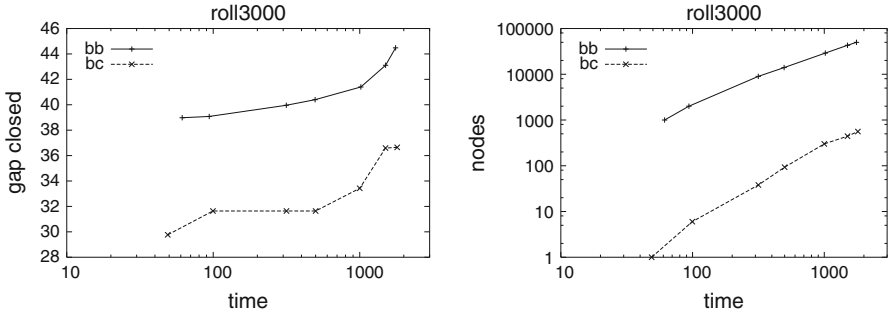

Fig. 7 Performance on roll3000

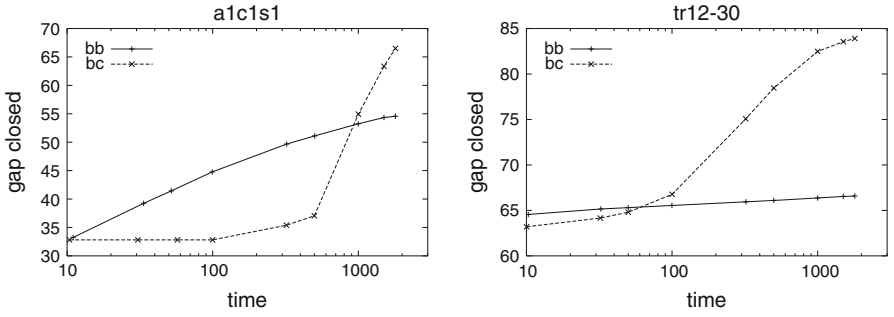

Fig. 8 Performance on a1c1s1 and tr12-30

The situation is more confused for the remaining three problems. As can be seen in Fig. 8, for *a1c1s1* and *tr12-30*, the gap closed by branch-and-cut is worse for some time, and then becomes better towards the 30 min mark. In the second problem, it is clear early on (from the slope of the gap closed versus time curve) that branch-and-cut will outpace branch-and-bound eventually, but not so in the case of *a1c1s1*.

The above experiment suggests that branch-and-cut with rank-1 GMI cuts generated at all nodes of the search tree may be competitive with branch-and-bound for some problems from MIPLIB 2003, at least with the CPLEX 11.2 MIP solver, and starting from the same root node. However, it is not clear what will happen once we allow CPLEX to generate its own (potentially high rank cuts) at the root and also employ dynamic search during branching. In our second experiment, we consider the problems in the first experiment and analyze the quality of bounds obtained by the rank-1 GMI cuts generated at the root by our heuristics. In Table 6, we give the number of nodes (rounded to the nearest hundred) and time CPLEX—with default settings—takes in order to obtain the same bound as that obtained by DEF without branching. For only three problems (*timtab1*, *timtab2*, *nsrand-ipx*) is the time we take to generate our bound competitive with the time taken by CPLEX. For *timtab1*, DEF takes 22 s before it terminates. CPLEX, with default settings, takes 53 s and explores 18,400 nodes before it obtains the same lower bound. As we know that GMI cuts are useful for this instance, we also compare the time taken if we let CPLEX aggressively generate all cuts ("set mip cuts all 2"). Henceforth, we refer to CPLEX with default

Table 6 Number of nodes required to recover bound using different CPLEX settings

Instance	DEF time	CPLEX nodes	Time	CPLEX+ nodes	Time
tr12-30	42	0	1	0	1
timtab1	22	18,300	53	10,100	35
timtab2	48	2,719,500	18,475	492,700	4,541
nsrand-ipx	1,545	153,300	5,372	24,400	745

settings simply as CPLEX, and CPLEX with aggressive cut generation as "CPLEX+". For *timtab2*, DEF is substantially better than CPLEX, or CPLEX+. On the other hand, for *tr12-30*, CPLEX takes less than a second (and performs no branching) in order to achieve the same bound DEF takes 42 s to attain.

From the first and second experiments, it is clear that for *timtab1* and *timtab2* it is likely that branch-and-cut with rank-1 GMI cuts generated by our heuristic will take less time to find the optimal solution than CPLEX branch-and-bound (or more precisely cut-and-branch). In our final experiment, we compare our branch-and-cut code with CPLEX and CPLEX+. We use the "DEF+BG100" setting (discussed in Table 4) at the root, and the "DEF-BG" setting at nodes of the branch-and-cut tree. As before, we turn off presolve and all solver cuts in our branch-and-cut enumeration.

We solve *timtab1* via branch-and-cut in 1,593 s and 7,500 nodes (this number is rounded to the nearest hundred) as compared to 3,689 s and 2,246,200 nodes with CPLEX (and 4,092 s and 2,495,400 nodes with CPLEX+). We also solve *timtab2* in about 332,000 s, i.e., in about 92 h, and with 265,900 nodes. This latter problem is not easy to solve. It was first solved in [14] by exploring about 17 million branch-and-bound nodes with CPLEX 9 using 2,745 h of CPU time and problem-specific cuts provided by Christian Liebchen, a formulator of the problem [29]. It was later solved in 2008 in 22 h on a standard PC by Liebchen and Swarat by a problem specific branch-and-cut method (see the MIPLIB 2003 web page). Therefore, our solution of *timtab2* is the first time it has been solved without problem specific techniques. Note that *timtab1* and *timtab2* both involve general integer variables and thus the techniques in [6] to generate globally valid GMI cuts cannot be employed. By solving a problem we mean that CPLEX stops the branch-and-bound process when it establishes a gap of at most 0.01% (the default value) between the incumbent integer solution and lower bound. The "optimal solution values" we obtain for the two problems above match those reported in MIPLIB 2003.

Though we explore 265,900 nodes in solving *timtab2*, the total number of rank-1 cuts added by our cut generator during the branch-and-cut process is only around 27,000 out of which 2,500 are added at the root node. In other words, fewer than one cut is added per node. In general, as the rank-1 cuts only use constraints of the original linear relaxation and do not use the branching constraints, we expect that there will be fewer violated rank-1 cuts deeper in the tree. We did not try out ideas such as imposing a *skip factor*—not generating a new round of cuts before processing a number of nodes of the search tree—as proposed for GMI cuts in [6] (though we do not know if CPLEX imposes this).

6 Conclusions

In this paper we presented a heuristic to generate rank-1 GMI cuts and showed that, for many MIPLIB problems, they can be used to obtain strong lower bounds, which are comparable to the bounds obtained by approximately optimizing over the MIR closure in [8] and [22]. Therefore, the rank-1 GMI cuts form a useful sub-class of the rank-1 MIR cuts. Further, we can often obtain these comparable bounds in a hundredth of the computing time.

We also used our heuristics to generate globally valid rank-1 GMI cuts in a branch-and-cut setting for problems which have general integer variables, and demonstrated that our heuristics are effective at finding violated cuts and significantly reducing the size of the branch-and-cut tree for some MIPLIB problems. A smaller branch-and-cut tree does not necessarily lead to less computing time. We solved *timtab1* and *timtab2*, two non-trivial problems from MIPLIB 2003, in significantly less time than the state-of-the-art MIP solver, CPLEX 11.2 (it cannot solve *timtab2* in 15 million nodes). However, it is clear from the branch-and-cut experiments in the previous section that much additional work needs to be performed before truly practical implementations of our heuristics can be obtained.

A major issue which needs additional study is the rank of the cuts to be used. Even though the current consensus seems to be that high rank cuts should not be used, it is not clear that only rank-1 cuts should be used. We feel that allowing the original constraint system to be augmented by very sparse cuts, especially bound implications (e.g, $x_i \geq 1$), before generating non-optimal bases and associated cuts may be a good idea. Further, our cut generation heuristics can be speeded up by standard techniques such as saving previously generated non-optimal bases and corresponding factorizations. Finally, we did not use common techniques such as limiting the density or coefficient magnitude variation of tableau rows to be used to generate cuts, nor did we perform any cut management, such as deletion of inactive cuts.

One criticism of our approach could be that they involve the generation of a large number of GMI cuts read from many tableaus. As the generation of GMI cuts in floating-point arithmetic is known to be error-prone (see Margot [32]) and can lead to invalid cuts, the use of a large number of tableaus in our method obviously increases the likelihood of invalid cuts. In [18], along with Cook and Fukasawa, we show that numerically safe GMI cuts can be generated in floating-point arithmetic. The techniques described in [18] can directly be incorporated in the heuristic presented in this article to make all generated cuts safe. Further, using the potentially weaker numerically safe cuts does not lead to a significant loss in the quality of lower bounds obtained vis-a-vis the unsafe cuts, nor is there a noticeable increase in the density of the safe cuts, for the MIPLIB problems tested in [18]. Based on these results, we are convinced that even if some of the cuts generated in the experiments in this paper turn out to be invalid, it is likely that if these invalid cuts were replaced by numerically safe GMI cuts, there would be no significant change in lower bounds.

An interesting extension of the work in this paper we are currently exploring is in the context of nonlinear programming problems which have nonlinear objectives/constraints in addition to many linear constraints. We could consider a solution of the entire system of constraints, and then search for bases of the sub-system of linear

constraints and generate GMI cuts based on these bases. For *ibienst1*, a MINLP with a convex quadratic objective and linear constraints available in Hans Mittelmann's library MIQPlib [34], we can obtain GMI cuts in this manner and a resulting lower bound in 3 s which CPLEX takes about 30 s and more than 100 nodes to attain.

Acknowledgments We would like to thank the referees for useful comments. We also thank Pierre Bonami for information on using the CglLandP cut generator.

References

1. Achterberg, T., Kock, T., Martin, A.: MIPLIB 2003. Oper. Res. Lett. **34**(4), 361–372 (2006)
2. Andersen, K., Cornuéjols, G., Li, Y.: Reduce-and-split cuts: improving the performance of mixed integer gomory cuts. Manage. Sci. **51**, 1720–1732 (2005)
3. Applegate, D., Cook, W., Dash, S., Mevenkamp, M.: QSopt linear programming solver. http://www.isye.gatech.edu/~wcook/qsopt (2004)
4. Balas, E., Bonami, P.: Generating lift-and-project cuts from the LP simplex tableau: open source implementation and testing of new variants. Math. Program. Comput. **1**, 165–200 (2010)
5. Balas, E., Ceria, S., Cornuejols, G.: A lift-and-project cutting plane algorithm for mixed 0–1 programs. Math. Program. **58**, 295–324 (1993)
6. Balas, E., Ceria, S., Cornuejols, G., Natraj, N.: Gomory cuts revisited. Oper. Res. Lett. **19**, 1–9 (1996)
7. Balas, E., Perregaard, M.: A precise correspondence between lift-and-project cuts, simple disjunctive cuts, and mixed integer Gomory cuts for 0–1 programming. Math. Program. **94**, 221–245 (2003)
8. Balas, E., Saxena, A.: Optimizing over the split closure. Math. Program. **113**, 219–240 (2008)
9. Bonami, P.: CglLandP. http://project.coin-or.org/Cgl/wiki/CglLandP
10. Bonami, P., Minoux, M.: Using rank-1 lift-and-project closures to generate cuts for 0–1 MIPs, a computational investigation. Discret. Optim. **2**, 288–307 (2005)
11. Bixby, R., Gu, Z., Rothberg, E., Wunderling, R.: The sharpest cut: the impact of Manfred Padberg and his work, chap. 18 (mixed-integer programming: a progress report), pp. 309–323. MPS-SIAM Series on Optimization (2004)
12. Bixby, R.E., Ceria, S., McZeal, C.M., Savelsbergh, M.W.P.: An updated mixed integer programming library: Miplib 3.0. Optima **58**, 12–15 (1998)
13. Bixby, R.E., Fenelon, M., Gu, Z., Rothberg, E., Wunderling, R.: Mip: theory and practice—closing the gap. In: System modelling and optimization, pp. 19–50 (1999)
14. Bussieck, M.R., Ferris, M.C., Meeraus, A.: Grid-enabled optimization with GAMS. INFORMS J. Comput. **21**(3), 349–362 (2009)
15. Caprara, A., Letchford, A.: On the separation of split cuts and related inequalities. Math. Program. **94**, 279–294 (2003)
16. Ceria, S., Cornuéjols, G., Dawande, M.: Combining and strengthening Gomory cuts. In: Balas, E., Clausen, J. (eds) Proceedings of IPCO 1995. Lect. Notes Comput. Sci. **920**, 438–451 (1995)
17. COIN-OR, The computational infrastructure for operations research project. http://www.coin-or.org
18. Cook, W., Dash, S., Goycoolea, M., Fukasawa, R.: Numerically safe Gomory mixed-integer cuts. INFORMS J. Comput. **21**, 641–649 (2009)
19. Cornuéjols, G., Li, Y.: Elementary closures for integer programs. Oper. Res. Lett. **28**, 1–8 (2001)
20. Dash, S., Günlük, O.: On the strength of Gomory mixed-integer cuts as group cuts. Math. Program. **115**, 387–407 (2008)
21. Dash, S., Goycoolea, M., Günlük, O.: Two-step mir inequalities for mixed-integer programs. INFORMS J. Comput. **22**, 236–249 (2010)
22. Dash, S., Günlük, O., Lodi, A.: MIR closures of polyhedral sets. Math. Program. **121**(1), 33–60 (2010)
23. Edmonds, J.: Matroids and the greedy algorithm. Math. Program. **1**(1), 127–136 (1971)
24. Fischetti, M., Lodi, A.: Optimizing over the first Chvátal closure. Math. Program. B **110**(1), 3–20 (2007)
25. Fischetti, M., Saturni, C.: Mixed integer cuts from cyclic groups. Math. Program. **109**, 27–53 (2007)
26. Fukasawa, R., Goycoolea, M.: On the exact separation of mixed-integer knapsack cuts. In: Fischetti, M., Williamson, D.P. (eds) Proceedings of IPCO 2007. Lect. Notes Comput. Sci. **4513**, 225–239 (2007)
27. Geoffrion, A.M., Graves, G.W.: Multicommodity distribution system design by Benders decomposition. Manage. Sci. **20**, 822–844 (1974)

28. Gomory, R.E.: An algorithm for the mixed integer problem. RM-2597, The Rand Corporation (1960)
29. Liebchen, C., Moehring, R.H.: Information on the MIPLIB's timetab-instances. Technical Report 2003/49, Department of Mathematics, Technical University Berlin (2003)
30. Marchand, H., Wolsey, L.A.: Aggregation and mixed integer rounding to solve MIPs. Oper. Res. **49**, 363–371 (2001)
31. Nemhauser, G.L., Wolsey, L.A.: Integer and combinatorial optimization. Wiley, New York (1988)
32. Margot, F.: Testing cut generators for mixed-integer linear programming. Math. Program. Comput. **1**, 69–95 (2009)
33. Markowitz, H.M.: The elimination from of inverse and its applications to linear programming. Manage. Sci. **3**, 255–269 (1957)
34. Mittelmann, H.: MIQPlib. http://plato.asu.edu/ftp/miqp/
35. Pinar, A., Chow, E., Pothen, A.: Combinatorial algorithms for computing column space bases that have sparse inverses. Electron. Trans. Numer. Anal. **22**, 122–145 (2006)
36. Suhl, U.H., Suhl, L.M.: Computing sparse LU factorizations for large-scale linear programming bases. ORSA J. Comput. **2**(4), 325–335 (1990)